Aerial Suspended Cargo Delivery through Reinforcement Learning

Aleksandra Faust* Ivana Palunko† Patricio Cruz‡ Rafael Fierro‡

Lydia Tapia*

August 7, 2013

Adaptive Motion Planning Research Group Technical Report TR13-001

Abstract

Cargo-bearing Unmanned aerial vehicles (UAVs) have tremendous potential to assist humans in food, medicine, and supply deliveries. For time-critical cargo delivery tasks, UAVs need to be able to navigate their environments and deliver suspended payloads with bounded load displacement. As a constraint balancing task for joint UAV-suspended load system dynamics, this task poses a challenge. This article presents a reinforcement learning approach to aerial cargo delivery tasks in environments with static obstacles. We first learn a minimal residual oscillations task policy in obstacle-free environments that find trajectories with minimized residual load displacement with a specifically designed feature vector for value function approximation. With insights of learning from the cargo delivery problem, we define a set of formal criteria for class of robotics problems where learning can occur in a simplified problem space and transfer to a broader problem space. Exploiting this property, we create a path tracking method that suppresses load displacement. As an extension to tasks in environments with static obstacles where the load displacement needs to be bounded throughout the trajectory, sampling-based motion planning generates collision-free paths. Next, a reinforcement learning agent transforms these paths into trajectories that maintain the bound on the load displacement while following the collision-free path in a timely manner. We verify the approach both in simulation and in experiments on quadrotor with suspended load.

1 Introduction

Unmanned aerial vehicles (UAVs) show potential for use in remote sensing, transportation, and search and rescue missions [10]. One such UAV, a quadrotor, is an ideal candidate for autonomous cargo delivery due to its trait of high maneuverability, vertical takeoff and landing, single-point hover, and ability to carry loads 50% to 100% of their body weight. For example, cargoes may consist of food and supply delivery in disaster struck areas, patient transport, or spacecraft landing. The four rotor blades of a quadrotor

*Department of Computer Science, University of New Mexico, Albuquerque, NM 87131, {afaust, tapia}@cs.unm.edu

†Electrical Engineering and Computing, University of Zagreb, Zagreb, Croatia, ivana.palunko@fer.hr

‡Department of Electrical and Computer Engineering, University of New Mexico, Albuquerque, NM, 87131-0001, {pcruzec, rfierro}@ece.unm.edu

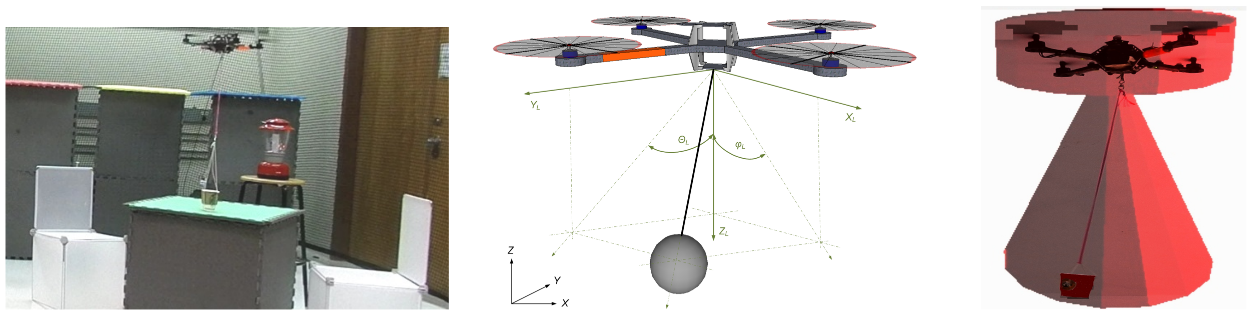
make them easier to maneuver than helicopters. However, they are still inherently unstable systems with complicated nonlinear dynamics. The addition of a suspended load further complicates the systems' dynamics, posing a significant control challenge. Planning motions that control the load position is difficult, so automated learning methods are necessary for mission safety and success.

Recent research has begun to develop control policies for mini UAVs, including approaches that incorporate learning [10]. Learning system dynamics model parametrization has been successful for adaptation to changes in aerodynamics conditions and system calibration [18, 24]. Other approaches, such as iterative learning methods for policy development, have been shown to be effective for aggressive maneuvers [13]. Another learning method, expectation-maximization, has been applied to the problem of quadrotor trajectory tracking with a target trajectory and a linear model [23]. Even the task of suspended load delivery has been addressed for UAVs [6, 21, 26, 3]. Our recent work used reinforcement learning (RL) in order to generate swing-free trajectories for quadrotors between two pre-defined, obstacle-free waypoints [6]. RL provided several advantages over previous work with dynamic programming [20]: a single learning phase leading to the generation of multiple trajectories, better compensation for accumulated error resulting from model approximation, and lack of knowledge of the detailed system dynamics. A similar problem, suspended load trajectory tracking has been performed with both RL [21] and model predictive control [26]. However, both of these methods require a pre-defined trajectory and do not automatically handle planning in environments with obstacles.

To accomplish task learning for a rotorcraft with suspended load, we rely on approximate value iteration [4] with a specifically designed feature vector for value function approximation. Transfer between tasks in different state and action spaces can be achieved using a behavior transfer function that transfers the value function to the new domain [27]. We transfer the learned value function to tasks with state and action space supersets and changed dynamics. We find sufficient characteristics of the target tasks for learning transfer to occur successfully. The direct transfer of the value function is sufficient and requires no further learning. Another approach examines action transfer between the tasks, learning the optimal policy and transferring only the most relevant actions from the optimal policy [25]. In obstacle-free spaces, we take the opposite approach; to save computational time, we learn a sub-optimal policy on a subset of actions, and transfer it to the expanded action space to produce a more refined plan. When creating a trajectory that tracks a path, we work with a most relevant subset of the expanded action space. Partial policy learning for fixed start and goal states, manage state space complexity by focusing on states that are more likely to be encountered [17]. We are interested in finding minimal residual oscillations trajectories from different start states, but we do have a single goal state. Thus, all trajectories will pass near the goal state, and we learn the partial policy only in the vicinity of the goal state. Then, we may apply it to any start state.

Motion planning methods, which define a valid, collision-free path for a robot, are often solved in *configuration space (C_{space})*, the space of all valid configurations. The valid, collision-free path lies in the collision-free portion of C_{space}, C_{free}. Many planning methods work by either learning and approximating the topology of C_{space}, e.g., PRMs [8], or by traversing a continuous path in C_{free}, e.g., RRT and EST methods [12, 7, 9]. One primary difference between PRMs and RRT/EST methods is that PRMs were designed to learn C_{free} topology once and then use this knowledge to solve multiple planning queries, and RRTs/ESTs expand from a single start and/or goal position for a single planning query. Recently, a modified PRM was shown to work in environments that were noisily modeled [14].

For the problem of suspended load control, we apply RL to automatically generate, test, and follow swing-free trajectories for a quadrotor. RL is integrated into both the motion planning and trajectory generation for a rotorcraft robot equipped with a suspended load and in an environment with static obstacles. In [6], we showed that RL could be used to control load displacement at the arrival to the goal state. Here, the agent is placed in a larger context of time-sensitive cargo delivery tasks. In this class of applications, the payload should be delivered free of collision as soon as possible, possibly tracking a reference path, while bounded load displacement is maintained throughout the trajectory. Beyond aerial robotics, the methods



(a) Experimental system (b) Position definition (c) Geometric model

Figure 1: Quadrotor carrying a suspended load.

presented here are applicable to any constraint balancing task posed on a dynamical system, because aerial cargo delivery problem is such a task.

The problem of learning controls for minimal residual oscillations falls in a class of learning problems where learning in one space and transferring the learned policy to a variety of spaces. For example, once the value function is learned, we demonstrate its use to generate any number of trajectories for the same payload. Relying on Lyapunov stability theory, we find sufficient criteria to allow the transfer of the learned, inferred policy to a variety of simulators, state, and action spaces. This allows us to implement path tracking with reduced load displacement by using the learned policy for minimal residual oscillations, and by restricting the action space at each step to maintain proximity to the reference trajectory, identified through planning. We also demonstrate learning state value function in 2-dimensional action space, and using it to generate altitude-changing trajectories when the value function is applied in 3-dimensional action space.

The result and contribution of this work is fully automated software system that plans and creates trajectories in an obstacle-laden environment for aerial cargo delivery. The RL agent presented, to the best of our knowledge, is currently the only reinforcement learning agent for creating trajectories with minimal residual oscillations for a suspended load bearing UAV, and we describe it in detail here. Further, to develop the automated system, we develop several novel methods: a reinforcement learning path tracking agent that reduces the load displacement, a framework for trajectory generation with constraints that avoids obstacles, reinforcement learning agent's integration with sampling-based path planning, and development of an efficient rigid body model to represent a quadrotor carrying a suspended load. We test the proposed methodology both in simulation and experimentally (Figure 1a).

2 Preliminaries

This article is concerned with joint UAV, suspended load systems. The load is suspended from the UAV's center of mass with an inelastic cable. A ball joint models the connection between the UAV's body and the suspension cable. We now define several terms that are used in this paper extensively: load displacement, swing, swing-free trajectory, a trajectory with minimal residual oscillations, and cargo delivery.

Definition 2.1. *(Load displacement or swing): at time t, is load's position in polar coordinates $\boldsymbol{\eta}(t) = [\phi(t)\ \theta(t)]^T$ with the origin in quadrotor's center of the mass (see Figure 1b).*

Definition 2.2. *(Minimal residual oscillations trajectory): A trajectory of duration T is a* minimal residual oscillations *trajectory if for a given constant $\epsilon > 0$ there is a time $0 \leq t_1 \leq T$, such that for all $t \geq t_1$, the load displacement is bounded with ϵ ($\|\boldsymbol{\eta}(t)\| < \epsilon$).*

Definition 2.3. *(Bounded load displacement or Swing-free trajectory): A trajectory is a* swing-free *trajectory if it is the minimal residual oscillation trajectory for time $t_1 = 0$, in other words if the load displacement is bounded by a given constant throughout the entire trajectory ($\|\boldsymbol{\eta}(t)\| < \epsilon,\ t \geq 0$).*

Now, we present problem formulation for two aerial cargo delivery tasks, which specify both the trajectory characteristic, task completion criteria, and dynamical constraints on the system.

Problem 2.1. *(Minimal residual oscillations task): Given start and goal positional states $\boldsymbol{x_s}, \boldsymbol{x_g} \in C_{free}$, and a swing-constraint ϵ, find a minimal residual oscillations trajectory $\boldsymbol{x} = \boldsymbol{\tau}(t)$ from $\boldsymbol{x_s}$ to $\boldsymbol{x_g}$ for a holonomic UAV carrying a suspended load. The task is complete when the system comes to rest at the goal state: for some ϵ_d, ϵ_v, at time t_g $\|\boldsymbol{\tau}(t_g) - \boldsymbol{x_g}\| \leq \epsilon_d$ and $\|\dot{\boldsymbol{\tau}}(t_g)\| \leq \epsilon_v$. The dynamical constraints of the system are the bounds on the acceleration vector $\boldsymbol{c_l} \leq \boldsymbol{a} \leq \boldsymbol{c_u}$.*

In contrast, aerial cargo delivery task requires bounded load displacement throughout the trajectory.

Problem 2.2. *(Cargo delivery task): Given start and goal positional states $\boldsymbol{x_s}, \boldsymbol{x_g} \in C_{free}$, and a swing-constraint ϵ, find a swing-free trajectory $x = \tau(t)$ from $\boldsymbol{x} = \boldsymbol{\tau}(t)$ from $\boldsymbol{x_s}$ to $\boldsymbol{x_g}$ for a holonomic UAV carrying a suspended load. The task is complete when the system comes to rest at the goal state: for some ϵ_d, ϵ_v, at time t_g $\|\boldsymbol{\tau}(t_g) - \boldsymbol{x_g}\| \leq \epsilon_d$ and $\|\dot{\boldsymbol{\tau}}(t_g)\| \leq \epsilon_v$. The dynamical constraints of the system are the bounds on the acceleration vector $\boldsymbol{c_l} \leq \boldsymbol{a} \leq \boldsymbol{c_u}$.*

3 Methods

Our goal is to develop a fully automated agent that calculates aerial cargo delivery (Problem 2.2) trajectories in environments with static obstacles. The trajectory must be collision-free, the load displacement must be bounded, and the UAV must arrive at the given goal coordinates. Within these constraints, the task needs to complete in the shortest time given the physical limitations of the system combined with the task constraints.

Figure 2 presents the proposed architecture. We learn the policy that reduces load displacement separately from the space geometry, and then combine them to generate trajectories with both characteristics. The swing-free policy performs minimal residual oscillations task (Problem 2.1) and is described in Section 3.1. Once the policy is learned, it can be used with varying state and action spaces to refine system performance according to the task specifications. Based on Lyapunov theory, Section 3.2 discusses the sufficient criteria for these modifications. In Section 3.3, we adapt the policy to track a reference path, by modifying the action space at every step. To enable obstacle-avoidance, we use PRMs to generate a collision-free path. Section 3.4 describes the geometric model of the cargo-bearing UAV that we use for efficient collision detection. After the policy is learned and a roadmap constructed, we generate trajectories. For given start and goal coordinates, the PRM query calculates a collision-free path between them. A trajectory along the path that maintains the acceptable load displacement is calculated using the method described in Section 3.5.

The architecture has two clear phases: learning and planning. The learning for a particular payload is performed once, and used many times in any environment. The problem geometry is learned, and PRMs are constructed, once for a given environment and maximum allowed load displacement. When constructed, roadmaps can be used for multiple queries in the environment for tasks requiring the same or smaller load displacement. The distinct learning phases and the ability to reuse both, the policy and the roadmap, are desirable and of practical use, because the policy learning and roadmap construction are time consuming.

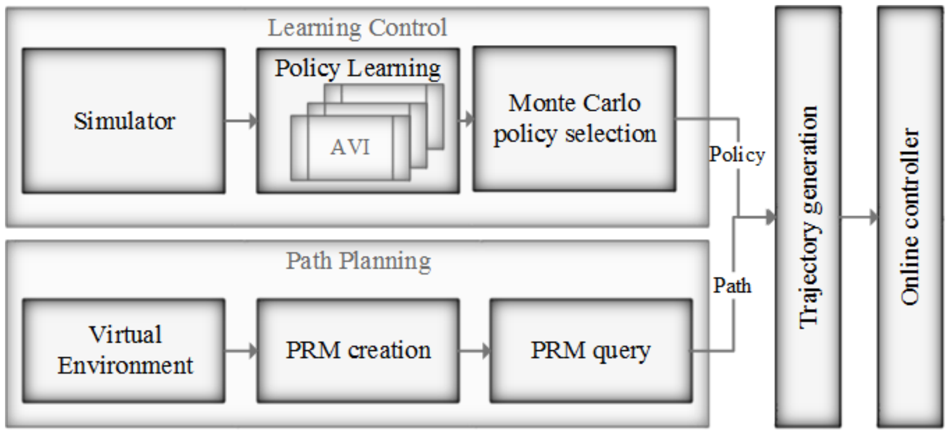


Figure 2: Automated aerial cargo delivery software architecture.

3.1 Learning minimal residual oscillations policy

In this section, our goal is to find fast trajectories with minimal residual oscillations for rotorcraft aerial robots carrying suspended loads in obstacle-free environments. We assume that we know the goal state of the vehicle; the initial state may be arbitrary. Furthermore, we assume that we have a *black box* system's simulator (or a generative model) available, but our algorithm makes no assumptions about the system's dynamics.

The approximate value iteration algorithm (AVI) [4] produces an approximate solution to a Markov Decision Process (MDP) defined with(S, A, D, R), in continuous state spaces with a discrete action set. A linearly parametrized feature vector approximates the value function. It is in an expectation-maximization (EM) algorithm that relies on a sampling of the state space transitions, an estimation of the state value function using the Bellman equation [2], and a linear regression to find the parameters that minimize the least squared error.

In our implementation, the MDP state space S is vector space $\boldsymbol{s} = [x\ y\ z\ \dot{x}\ \dot{y}\ \dot{z}\ \phi_L\ \theta_L\ \dot{\phi}_L\ \dot{\theta}_L\ ]^T$. Vector $\boldsymbol{p} = [x\ y\ z]^T$ is the position of the vehicle's center of mass, relative to the goal state. The vehicle's linear velocity is vector $\boldsymbol{v} = [\dot{x}\ \dot{y}\ \dot{z}]^T$. Vector $\boldsymbol{\eta} = [\phi_L\ \theta_L]^T$ represents the angles that the suspension cable projections onto xz and yz planes form with the z-axis (see Figure 1b). The vector of the load's angular velocities is $\dot{\boldsymbol{\eta}}_{\boldsymbol{L}} = [\dot{\phi}_L\ \dot{\theta}_L]^T$. L is the length of the suspension cable. Since L is constant in this work, it will be omitted. To simplify, we also refer to the state as $\boldsymbol{s} = [\boldsymbol{p}\ \boldsymbol{v}\ \boldsymbol{\eta}\ \dot{\boldsymbol{\eta}}]^T$ when we do not need to differentiate between particular dimensions in $\boldsymbol{s}$. The action space, A is a set of linear acceleration vectors $\boldsymbol{a} = [\ddot{x} \quad \ddot{y} \quad \ddot{z}]^T$ discretized using equidistant steps centered around zero acceleration.

The reward function, R, penalizes the distance from the goal state, and the load displacement angle. It also penalizes the negative z coordinate to provide a bounding box and enforce that the vehicle must stay above the ground. Lastly, the agent is rewarded when it reaches the goal. The reward function $R(\boldsymbol{s}) = \boldsymbol{c}^T\boldsymbol{r}(\boldsymbol{s})$ is a linear combination of basis rewards $\boldsymbol{r}(\boldsymbol{s}) = [r_1(\boldsymbol{s})\ r_2(\boldsymbol{s})\ r_3(\boldsymbol{s})]^T$, weighted with vector $\boldsymbol{c} = [c_1\ c_2\ c_3]^T$, for some constants a_1 and a_2, where:

$$r_1(\boldsymbol{s}) = -\|\boldsymbol{p}\|^2 \qquad r_2(\boldsymbol{s}) = \begin{cases} a_1 & \|\boldsymbol{F}(\boldsymbol{s})\| < \epsilon \\ -\|\boldsymbol{\eta}\|^2 & \text{otherwise} \end{cases} \qquad r_3(\boldsymbol{s}) = \begin{cases} -a_2 & z < 0 \\ 0 & z \geq 0 \end{cases}$$

To obtain the state transition function samples $\boldsymbol{D}(\boldsymbol{s_0}, \boldsymbol{a}) = \boldsymbol{s}$, we rely on a simplified model of the quadrotor-load system, where the quadrotor is represented by a holonomic model of a UAV [10, 6, 19].

The simulator returns the next system state $\boldsymbol{s} = [\boldsymbol{p}\ \boldsymbol{v}\ \boldsymbol{\eta}\ \dot{\boldsymbol{\eta}}]$ when an action $\boldsymbol{a}$ is applied to a state $\boldsymbol{s_0} = [\boldsymbol{p_0}\ \boldsymbol{v_0}\ \boldsymbol{\eta_0}\ \dot{\boldsymbol{\eta}}_\mathbf{0}]$. Equations

$$\begin{aligned}
&\boldsymbol{v} = \boldsymbol{v_0} + \triangle t\boldsymbol{a}; \quad \boldsymbol{p} = \boldsymbol{p_0} + \triangle t\boldsymbol{v_0} + \frac{\triangle t^2}{2}\boldsymbol{a} \\
&\dot{\boldsymbol{\eta}} = \dot{\boldsymbol{\eta}}_\mathbf{0} + \triangle t\ddot{\boldsymbol{\eta}}; \quad \boldsymbol{\eta} = \boldsymbol{\eta_0} + \triangle t\dot{\boldsymbol{\eta}}_\mathbf{0} + \frac{\triangle t^2}{2}\ddot{\boldsymbol{\eta}}, \text{ where} \\
&\ddot{\boldsymbol{\eta}} = \begin{bmatrix} \sin\theta_0\sin\phi_0 & -\cos\phi_0 & L^{-1}\cos\theta_0\sin\phi_0 \\ -\cos\theta_0\cos\phi_0 & 0 & L^{-1}\cos\phi_0\sin\theta_0 \end{bmatrix}(\boldsymbol{a} - \boldsymbol{g'})
\end{aligned} \tag{1}$$

describe the simulator. A vector $\boldsymbol{g'} = [0\ 0\ g]^T$ represents the gravity force vector, L is the length of the suspension cable, and $\triangle t$ is the duration of the time step.

The state value function V is approximated with a weighted feature vector $\boldsymbol{F}(\boldsymbol{s})$. The feature vector chosen for this problem consists of four basis functions: squares of the vehicles distance to the goal, its velocity magnitude, and load displacement and velocity magnitude:

$$V(\boldsymbol{s}) = \boldsymbol{\Theta}^T\boldsymbol{F}(\boldsymbol{s}), \quad \boldsymbol{F}(\boldsymbol{s}) = [\|\boldsymbol{p}\|^2 \quad \|\boldsymbol{v}\|^2 \quad \|\boldsymbol{\eta}\|^2 \quad \|\dot{\boldsymbol{\eta}}\|^2]^T \tag{2}$$

where $\boldsymbol{\Theta} \in \mathbb{R}^4$.

To learn the approximation of the state value function, AVI learns parametrization $\boldsymbol{\Theta}$. Starting with an arbitrary vector $\boldsymbol{\Theta}$, in each iteration, the state space is randomly sampled to produce a set of state samples M. The samples are uniformly, randomly drawn from a 10-dimensional interval centered in the goal state at equilibrium. A new estimate of the state value function is calculated according to $V(\boldsymbol{s}) = R(\boldsymbol{s}) + \gamma \max_{\boldsymbol{a}} \boldsymbol{\Theta}^T\boldsymbol{F} \circ \boldsymbol{D}(\boldsymbol{s}, \boldsymbol{a})$ for all samples $\boldsymbol{s} \in M$. $0 < \gamma < 1$ is a discount factor, and $\boldsymbol{D}(\boldsymbol{s}, \boldsymbol{a})$ is sampled transition when action $\boldsymbol{a}$ is applied to state $\boldsymbol{s}$. A linear regression then finds a new value of $\boldsymbol{\Theta}$ that fits the calculated estimates $V(\boldsymbol{s})$ into quadratic form $\boldsymbol{\Theta}^T\boldsymbol{F}(\boldsymbol{s})$. The process repeats until a maximum number of iterations is performed.

3.2 Minimal residual oscillations trajectory generation

An approximated value function, V, induces a greedy policy $\boldsymbol{\pi} : S \to A$ that is used to generate the trajectory and control the vehicle,

$$\boldsymbol{\pi}(\boldsymbol{s}) = \underset{\boldsymbol{a} \in A}{\operatorname{argmin}}(\boldsymbol{\Theta}^T\boldsymbol{F} \circ \boldsymbol{D}(\boldsymbol{s}, \boldsymbol{a})), \tag{3}$$

where $\boldsymbol{D}$ is the state transition function described in (1) resulting action moves the system to the state associated with the highest estimated value. The algorithm starts with the initial state. Then it finds an action according to (3). The action is used to transition to the next state. The process repeats until the goal is reached or the trajectory exceeds a maximum number of steps.

The Proposition 3.1 gives sufficient conditions that the value function approximation, action state space and system dynamics need to meet to guarantee a plan that leads to the goal state.

Proposition 3.1. *Let $\boldsymbol{s_g}$ be the goal state. If:*

1. *All components of vector $\boldsymbol{\Theta}$ are negative, $\Theta_i < 0$, for $\forall i \in \{1, 2, 3, 4\}$,*
2. *Action space, A, allows transitions to a higher-valued state, $\forall \boldsymbol{s} \in S\backslash\{\boldsymbol{s_g}\}, \exists \boldsymbol{a} \in A$ that $V(\boldsymbol{\pi}_A(\boldsymbol{s})) > V(\boldsymbol{s})$, and*
3. *$\boldsymbol{s_g}$ is global maximum of function V,*

then the $\boldsymbol{s_g}$ is an asymptotically stable point. Coincidentally, for an arbitrary start state $\boldsymbol{s} \in S$, greedy policy (3) *with respect to V defined in* (2) *and A, leads to the goal state $\boldsymbol{s_g}$. In other words, $\forall \boldsymbol{s} \in S, \exists n, \boldsymbol{\pi}^n(\boldsymbol{s}) = \boldsymbol{s_g}$.*

Proof. To show that s_q is an asymptotically stable point, we need to find a discrete time Lyapunov control function $W(\boldsymbol{s})$, such that

1. $W(\boldsymbol{s}(k)) > 0, \quad$ for $\forall \boldsymbol{s}(k) \neq 0$
2. $W(\boldsymbol{s_g}) = 0,$
3. $\triangle W(\boldsymbol{s}(k)) = W(\boldsymbol{s}(k+1)) - W(\boldsymbol{s}(k)) < 0, \quad \forall k \geq 0$
4. $\triangle W(\boldsymbol{s}_g) = 0, \quad$ where $\boldsymbol{s_g} = [0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]^T$

Let $W(\boldsymbol{s}) = -V(\boldsymbol{s}) = -\boldsymbol{\Theta}^T[\|\boldsymbol{p}\|^2\ \|(\boldsymbol{v})\|^2\ \|\boldsymbol{\eta}\|^2\ \|\dot{\boldsymbol{\eta}}\|^2]$. Then $W(\boldsymbol{0}) = 0$, and for all $\boldsymbol{s} \neq \boldsymbol{s_g}$, $W(\boldsymbol{s}) > 0$, since $\Theta_i < 0$.

$\triangle W(\boldsymbol{s}(k)) = -(V(\boldsymbol{s}(k+1)) - V(\boldsymbol{s}(k))) < 0$ because of the assumption that for each state there is an action that takes the system to a state with a higher value. Lastly, since $V(\boldsymbol{s_g})$ is global maxima, $W(\boldsymbol{s_g})$ is global minima, and $\triangle W(\boldsymbol{s_g}) = -(V(\boldsymbol{s_g}) - V(\boldsymbol{s_g})) = 0$

Thus, W is a Lyapunov function with no constraints on $\boldsymbol{s}$, and is globally asymptotically stable. Therefore, any policy-following function W (or V) will lead the system to the unique equilibrium point. □

Proposition 3.1 connects the state value approximation with Lyapunov stability analysis theory. If V satisfies the criteria, the system is globally approximately stable, i.e. a policy generated under these criteria will drive the quadrotor-load system from any initial state $\boldsymbol{s}$ to the goal state $\boldsymbol{s_g} = \boldsymbol{0}$. We empirically show that the criteria is met. Proposition 3.1 requires all components of vector $\boldsymbol{\Theta}$ to be negative. As we will see in the 4.2, the empirical results show that is the case. These observations lead to several practical properties of the induced greedy policy that we will verify empirically:

- The policy is agnostic to the simulator used; the simulator defines the transition function, and along with the action space, defines the set of reachable states. Thus, as long as the conditions of Proposition 3.1 are met, we can switch the simulators we use. This means that we can train on a simple simulator and generate a trajectory on a more sophisticated model that would predict the system better.

- The policy can be learned on a state space subset that contains the goal state, and the resulting policy will work on the whole domain where the criteria above hold, i.e., where the value function doesn't have other maxima. We will show this property experimentally in Sections 4.2 and 4.2.2

- The induced greedy policy is robust to noise; as long as there is a transition to a state with a higher value, the action will be taken and the goal will be attained. Section 4.2 presents the empirical evidence for this property.

- The action space between learning and the trajectory generation can change, and the algorithm will still produce a trajectory to the goal state. For example, to save computational time, we can learn on a smaller, more coarse discretization of the action space to obtain the value function parameters, and generate a trajectory on a more refined action space which produces a smoother trajectory. We will demonstrate this property during the altitude changing flight experiment in Section 4.2.2. This property also allows us to create a path tracking algorithm in Section 3.3 by restricting the action space only to actions that maintain proximity to the reference trajectory.

Since we use an approximation to represent a value function and obtain an estimate iteratively, the question of algorithm convergence is twofold. First, the parameters that determine the value function must converge to a fixed point. Second, the fixed point of the approximator must be close to the true value function. Convergence of the algorithm is not guaranteed in the general case. Thus, we will show empirically that the approximator parameters stabilize. To show that the policy derived from a stabilized approximator is sound, we will examine the resulting trajectory. The trajectory needs to be with minimal residual oscillations at the arrival at the goal state, and be suitable for the system.

Thus far, we used reinforcement learning to learn the minimal residual oscillations task (Problem 2.1). The learning requires a feature vector, and a generative model of system dynamics, to come up with the feature vector parametrization. Then, in the distinct trajectory generation phase, using the same feature vector, and possibly different simulators, states, and action spaces, we create trajectories. The learning is done once for a particular payload, and the learned policy is used for any number of trajectories. Once the trajectory is created, it is passed to the lower-level vehicle controller.

3.3 Swing-free path tracking

To learn how to generate a trajectory that follows a reference path, while maintaining minimal residual oscillations, we develop a novel approach that takes advantage of findings from Section 3.2 by applying nonholomonic constraints on the system. In particular, we rely on ability to change the action space and still complete the task.

Let $\boldsymbol{P} = [\boldsymbol{r_1}, .., \ \boldsymbol{r_n}]$ be a reference path, given as the list of quadrotor's center of mass Cartesian coordinates in 3D space, and let $d_{\boldsymbol{P}}(\boldsymbol{s})$ be shortest Euclidean distance between the reference trajectory and the system's state $\boldsymbol{s} = [\boldsymbol{p} \ \boldsymbol{v} \ \boldsymbol{\eta} \ \dot{\boldsymbol{\eta}}]^T$:

$$d_{\boldsymbol{P}}(\boldsymbol{s}) = \min_i \|\boldsymbol{p} - \boldsymbol{r_i}\|. \tag{4}$$

Then we can formulate the swing-free path tracking task, as:

Problem 3.1. *(Swing-free path tracking task): Given reference path $\boldsymbol{P}$, and start and goal positional states $\boldsymbol{x_s}, \boldsymbol{x_g} \in \boldsymbol{P}$, find a minimal residual oscillations trajectory from $\boldsymbol{x_s}$to $\boldsymbol{x_g}$ that minimizes accumulated squared error: $\int_0^\infty (d_{\boldsymbol{P}}(\boldsymbol{s}))^2 \, dt$.*

To keep the system in the proximity of the reference path, we restrict the action space $A_{\boldsymbol{s}} \subset A$ to only the actions that transition the system in the proximity of the reference path, using proximity constant δ:

$$A_{\boldsymbol{s}} = \{\boldsymbol{a} \in A | d_{\boldsymbol{P}}(D(\boldsymbol{s}, \boldsymbol{a})) < \delta\}. \tag{5}$$

In the case $A_{\boldsymbol{s}} = \emptyset$, we use an alternative action subset that transitions the system to the m closest position to the reference trajectory,

$$A_{\boldsymbol{s}} = \operatorname*{argmin}_{\boldsymbol{a} \in A}{}^m (d_{\boldsymbol{P}}(D(\boldsymbol{s}, \boldsymbol{a}))), \tag{6}$$

where argmin^m denotes m smallest elements. We call m the candidate action set size constant. Admissible action set $A_{\boldsymbol{s}} \subset A$ represents a nonholonomic constraints on the system.

The action selection step, or the policy, becomes the search for action that transitions the system to the highest valued state chosen from the $A_{\boldsymbol{s}}$ subset,

$$\boldsymbol{\pi}(\boldsymbol{s}) = \operatorname*{argmin}_{\boldsymbol{a} \in A_{\boldsymbol{s}}} (\boldsymbol{\Theta}^T \boldsymbol{F} \circ \boldsymbol{D}(\boldsymbol{s}, \boldsymbol{a})). \tag{7}$$

The policy given in (7) ensures state transitions in the vicinity of the reference path $\boldsymbol{P}$. If it is not possible to transition the system within given ideal proximity δ, the algorithm selects m closest positions and selects an action that produces the best minimal residual oscillations characteristics upon transition (see Algorithm 1). Varying δ, which is the desired error margin, controls how close we desire the system to be to the reference path. Parameter m gives the weight to whether we prefer the proximity to the reference trajectory, or load displacement reduction. Choosing very small m and δ results in trajectories that are as close to the reference trajectory as the system dynamics allows, at the expense of the load swing. Larger values of the parameters allow more deviation from the reference trajectory, and better load displacement results.

Algorithm 1 Swing-free path tracking

```
Input: $\boldsymbol{s_0}$ start state, MDP $(S, A, D, R)$ state space,
Input: $\boldsymbol{\Theta}$, $\boldsymbol{F}(\boldsymbol{s})$,$max\_steps$
Input: $\boldsymbol{P}$ , $\delta, m$
Output: $trajectory$
1: $\boldsymbol{s} \leftarrow \boldsymbol{s_0}$
2: $trajectory \leftarrow empty$
3: while not(goal reached or $max\_steps$ reached) do
4:     $A_{\boldsymbol{s}} \leftarrow \{\boldsymbol{a} | d(\boldsymbol{s'}, \boldsymbol{P}) < \delta, \boldsymbol{s}' = D(\boldsymbol{s}, \boldsymbol{a}), \boldsymbol{a} \in A\}$
5:     if $A_{\boldsymbol{s}} == \emptyset$ then
6:         $A_{\boldsymbol{s}} \leftarrow \text{argmin}^m{}_{\boldsymbol{a} \in A}(d(D(\boldsymbol{s}, \boldsymbol{a}), P))$
7:     end if
8:     $\boldsymbol{a} \leftarrow \text{argmin}_{\boldsymbol{a} \in A_{\boldsymbol{s}}} \boldsymbol{\Theta}^T \boldsymbol{F} \circ \boldsymbol{D}(\boldsymbol{s}, \boldsymbol{a}))$
9:     add $(\boldsymbol{s}, \boldsymbol{a})$ to $trajectory$
10:    $\boldsymbol{s} \leftarrow D(\boldsymbol{s}, \boldsymbol{a})$
11: end while
12: return $trajectory$
```

3.4 UAV geometric model

In our method, we use PRMs to obtain a collision-free path. The PRMs require a geometric model of the physical space, and of a robot, to construct an approximate model of C_{space}. In order to simplify the robot model, we use its bounding volume. The selection of the robot model has implications for the computational cost of the collision detection [11]. The bounding volume can be represented with fewer degrees of freedom for simplicity, where certain joints are omitted. This has been done successfully for robot representations of molecular motion [15].

We find a geometric representation of the quadrotor UAV carrying a suspended load that assumes the load displacement is constrained within a given limit. The whole system is a 8-DoF system: two rigid bodies joined with a ball joint. Looking at the quadrotor's body, we can ignore yaw since the quadrotor is symmetrical along the z-axis. For small linear velocities, pitch and roll are negligible and we can ignore them as well. Thus the quadrotor's body can be modeled as a cylinder encompassing its entire body. This is a 3-DoF rigid body capable only of translation in the 3-dimensional physical space.

For the load, we need only to consider the set of its possible positions. We assume an inelastic suspension cable and fixed maximum load displacement angle. The set of all allowed load positions is a spherical sector that remains in fixed position to the quadrotor's body. This sector is contained in a right circular cone with an apex that connects to the quadrotor's body, and with its axis perpendicular to quadrotor's body. The relationship between the cone's aperture, ρ, and load displacement $\boldsymbol{\eta} = [\phi \ \theta]^T$ that need to be satisfied for collision-free trajectories is $\cos(\rho/2) > (1 + \tan^2 \phi + \tan^2 \theta)^{-0.5}$. Connecting the cylindrical model of the quadrotor body with the cone model of its load gives us the geometrical model of the quadrotor carrying a suspended load. Since the quadrotor's body stays orthogonal to the z-axis, the cylinder-cone model can be treated as a single rigid body with 3 degrees of freedom. Figure 1c shows the fit of the AscTec Hummingbird quadrotor carrying a suspended load fitting into the cylinder-cone bounding volume. The clearance surrounding the quadrotor's body area needs greater than the maximal path tracking error, $\max_t d_{\boldsymbol{P}}(\boldsymbol{s}(t))$. The model, will produce paths that leave enough clearance between the obstacles to accommodate for the maximum allowable load displacement.

3.5 Path planning and trajectory generation integration

Figure 2 shows the flowchart of the trajectory generation process. This method learns the dynamics of the system through AVI, as outlined in Section 3.1. Monte Carlo selection picks the best policy out of the multiple learning trials. Independently and concurrently, a planner learns the space geometry. When the system is queried with particular start and stop goals, the planner returns a collision-free path. AVI provides the policy. The trajectory generation module generates a swing-free trajectory along each edge in the path, using Algorithm 1 and the modified AVI policy. If the trajectory exceeds the maximum allowed load displacement, the edge is bisected, and the midpoint is inserted in the waypoint list (see Algorithm 2). The PRM paths in this case are a list of waypoints. The system comes to a stop in all waypoints and starts the next trajectory from the stop state. All swing-free trajectories are translated and concatenated to produce the final multi-waypoint, collision-free trajectory. This trajectory is sent to the baseline altitude controller that performs trajectory tracking [20]. The result is fully automated method that solves cargo delivery task (Problem 2.2).

Line 12 in Algorithm 2 creates a trajectory segment that tracks the PRM calculated path between adjacent nodes (w_i, w_{i+1}). The local planer determines the path geometry. Although any local planner can be used with PRMs, in this setup we choose a straight line local planner to construct the roadmap. Thus, the collision-free paths are line segments. Prior to creating a segment, we translate the coordinate system such that the goal state is in the origin. Upon trajectory segment calculation, the segment is translated so that it ends in w_{i+1}. The swing-free path tracking corresponds to Algorithm 1. When the reference path is a line segment with start in $\boldsymbol{s_0}$ and end in the origin, such as in this setup, the distance d calculation defined in (4), and needed to calculate the action space subsets (5), and (6), can be simplified and depends only the start state $\boldsymbol{s_0}$:

$$d_{\boldsymbol{s_0}}(\boldsymbol{s}) = \sqrt{\frac{\|\boldsymbol{s}\|^2\|\boldsymbol{s_0}\|^2 - (\boldsymbol{s}^T\boldsymbol{s_0})^2}{\|\boldsymbol{s_0}\|}}.$$

Algorithm 2 leads the system to the goal state. This is an implication of the asymptotic stability of the AVI algorithm. It means that the policy will produce trajectories starting in any initial state $\boldsymbol{s_0}$ and will come to rest at the origin. Because Algorithm 2 translates the coordinate system such that the next waypoint is always in the origin, the system will progress through waypoints until the end of the path is reached.

Algorithm 2 is also probabilistically complete, if a suitable trajectory between a pair of states exists, the probability of the algorithm finding it converges to one as the time approaches infinity [11]. The path finding through PRMs is probabilistically complete [8]. When a trajectory between two waypoints a and b is created, we check if it exceeds the maximum allowed load displacement, and in that case, add another waypoint c equidistant from waypoints a and b. Since c belongs to an obstacle-free path ab, paths ac and cb will be in C_{free} and half in length. The maximum load displacement in the trajectory depends on its length, as we will see in Section 4.2. Thus trajectories on ac and cb paths will have smaller load displacement than the ab trajectory. The algorithm repeats the bisecting process until the path is sufficiently small. This process is finite per edge in the path and thus probabilistically complete. Thus, the probability of Algorithm 2 finding a trajectory with bounded load displacement as time increases approaches 1, if a trajectory exists.

Algorithm 2 Collision-free cargo delivery trajectory generation

```
Input: $start, goal,$
Input: $space\_geometry, robot\_model,$
Input: $dynamics\_generative\_model, max\_swing$
Output: $trajectory$
 1: if $prm$ not created then
 2:   $prm.create(space\_geometry, robot\_model)$
 3: end if
 4: if $policy$ not created then
 5:   $policy \leftarrow avi.learn(dynamics\_generative\_model)$
 6: end if
 7: $trajectory \leftarrow []$
 8: $path \leftarrow prm.query(start, goal)$
 9: $currentStart \leftarrow path.pop()$
10: while not path.empty() do
11:   $distance \leftarrow currentStart - currentGoal$
12:   $tSegmeent \leftarrow avi.executeTrack(distance, policy)$
13:   if $tSegmeent.maxSwing > max\_swing$ then
14:     $midpoint \leftarrow \frac{currentStart+currentGoal}{2}$
15:     $path.push(midpoint)$
16:     $currentGoal \leftarrow midpoint$
17:   else
18:     $tSegment \leftarrow translate(tSegment, currentGoal)$
19:     $trajectory \leftarrow trajectory + tSegment$
20:     $currentStart \leftarrow currentGoal$
21:     $currentGoal \leftarrow path.pop()$
22:   end if
23: end while
24: return $trajectory$
```

4 Results

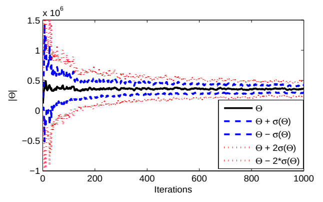
To evaluate the aerial cargo delivery software architecture, we first check each of the components separately, and then evaluate the architecture as a whole. Section 4.1 validates the learning and its convergence to a single policy. Section 4.2 evaluates policy for minimal residual oscillation trajectory generation and its performance under varying state spaces, action spaces, and system simulators, as Proposition 3.1 predicts. In Section 4.3, we address the quality of the swing-free tracking algorithm. After results for all components are presented, we verify the end-to-end process in Section 4.4.

All of the computations were performed on a single core of an Intel i9 system with 8GB of RAM, running Linux operating system. AVI and trajectory calculations were obtained with Matlab 2011. The experiments are performed using an AscTec quadrotor UAV carrying a small ball or a paper cup filled with water in a MARHES multi-aerial vehicle testbed [16]. This testbed and its real-time controller are described in detail in [20]. The testbed is equipped with a Vicon [28] high-precision motion capture system to collect the results of the vehicle and the load positioning during the experiments. The quadrotor is 36.5cm in diameter, weighs 353 g without a battery, and its payload is up to 350 g [1]. Meanwhile, the ball payload used in the experiments weighs 47 g and its suspension link length is 0.62 m. The suspended load is attached to the quadrotor at all times during the experiments. In experiments where the coffee cup is used, the weight of the payload is 100 g.

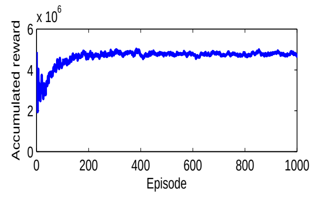
4.1 Learning minimal residual oscillations policy

To empirically verify the learning convergence, we run AVI in two configurations: 2D and 3D. Both configurations use the same discount parameter $\gamma < 1$ to ensure that the value function is finite. The configurations also share the simulator, described in (1). The 3D configuration trains the agent with a coarse three-dimensional action vector. Each direction of the linear acceleration is discretized in 13 steps, resulting in 13^3 total actions. In this phase of the algorithm, we are shaping the value function, and this level of coarseness is sufficient. AVI's approximation error decays exponentially with the number of iterations. A gradual increase in the sampling over iterations yields less error as the number of iterations increases [5]. Thus, we increase sampling linearly with the number of iterations in the 3D configuration.

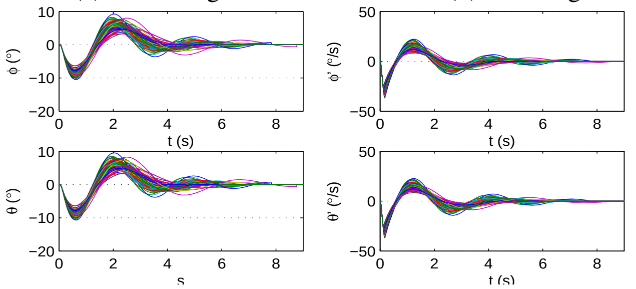
To assess the stability of the approximate value iteration, we ran the AVI 100 times, for 1000 iterations in the 3D configuration. Figure 3a shows the trend of the norm of value parameter vector $\boldsymbol{\Theta}$ with respect to L_2 norm. We can see that the $\|\boldsymbol{\Theta}\|$ stabilizes after about 200 iterations with the mean of 361170. The empirical results show that the algorithm is stable and produces a consistent policy over different trials. The mean value of $\boldsymbol{\Theta} = [-86290 \ -350350 \ -1430 \ -1160]^T$ has all negative components, which means that the assumption for Proposition 3.1 holds. To access learning progress, a trajectory generation episode was ran after every learning iteration. Figure 3b depicts the accumulated reward per episode, averaged over 100 trials. The accumulated reward converges after 200 iterations as well. Figure 3 depicts trajectories with the start state in $(-2, -2, 1)$ over 100 trials after 1000 learning iterations. Although there are slight variations in duration (see Figure 3c), all the trajectories are similar in shape and are consistent, giving us confidence that the AVI converges. The load initially lags behind as the vehicle accelerates (see Figure 3d), but then stabilizes to end in a minimal swing. We can also see that the swing is bounded throughout the trajectory, maintaining the displacement under $10°$ for the duration of the entire flight (see Figure 3d).



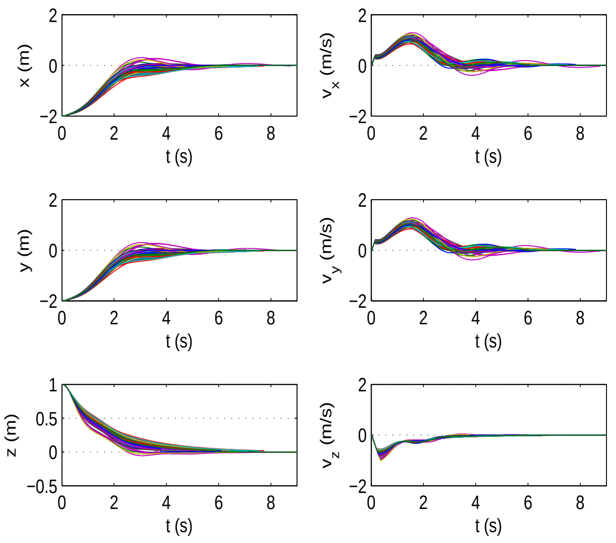
(a) Θ convergence



(b) Learning curve



(d) Load position and velocity



(c) Quadrotor position and velocity

Figure 3: Resulting trajectories from 100 policy trials. Trajectories starting at (-2, -2, 1) for each of the 100 trials of the vehicle (a), and its load (b) using 3D configuration for training and holonomic simulator with fine-grain action space for trajectory generation. Convergence of feature parameter vector $\boldsymbol{\Theta}$'s norm (c) and corresponding learning curve (d) over 1000 iterations averaged over 100 trials.

4.2 Minimal residual oscillations trajectory generation

In this section we evaluate effectiveness of the learned policy. We show the policy's viability in the expanded state and action spaces in simulation in Section 4.2.1. Section 4.2.2 assess the discrepancy between the load displacement predictions in simulation and encountered experimentally during the flight. The section also compares experimentally the trajectory created using the learned policy to two other methods: a cubic spline trajectory, which is a C^3-class trajectory without any pre-assumptions about the load swing, and, to a dynamic programming trajectory [20], an optimal trajectory for a fixed start position with respect to its MDP setup.

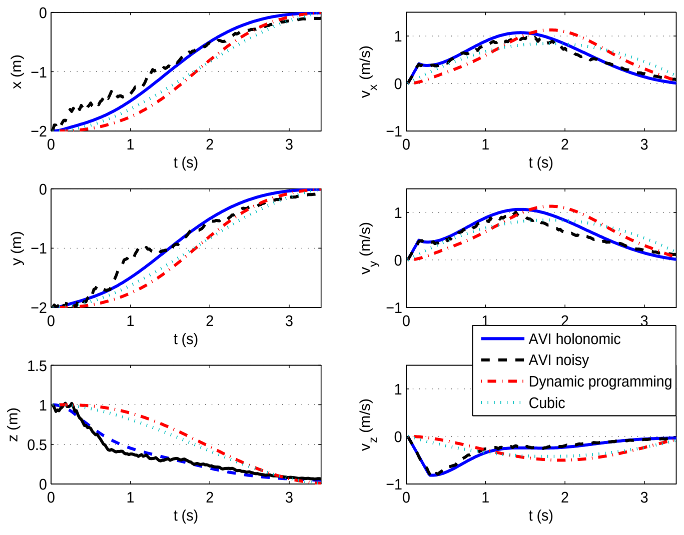
4.2.1 State and action space expansion

We assess the quality and robustness of a trained agent in simulation by generating trajectories from different distances for two different simulators. The first simulator is a generic holonomic aerial vehicle with a suspended load simulator, the same simulator we used in the learning phase. The second simulator is a stochastic holonomic aerial vehicle with suspended simulator that adds up to 5% uniform noise to the predicted state. Its intent is to simulate the inaccuracies and uncertainties of the physical hardware motion. We compare the performance of our learned, generated trajectories with model-based dynamic programming (DP) and cubic trajectories. The cubic and DP trajectories are generated using methods described in [20], but instead of relying on the full quadrotor-load system, we use the simplified model given by (1). The cubic and DP trajectories are of the same duration as corresponding learned trajectories. The agent is trained in 3D configuration. For trajectory generation, we use a fine-grain, discretized 3D action space $A = (-3 : 0.05 : 3)^3$. This action space is ten times per dimension finer, and contains 121^3 different actions. The trajectories were generated at 50Hz with a maximum duration of 15 seconds. All trajectories were generated and averaged over 100 trials. To assess how well a policy adapts to different starting positions, we choose two different fixed positions, (-2,-2, 1) and (-20,-20,15), and two variable positions. The variable positions are randomly drawn from between 4 and 5 meters, and within 1 meter from the goal state. The last position measures how well the agent performs within the sampling box. The rest of the positions are well outside of the sampling space used for the policy generation, and assess how well the method works for trajectories outside of the sampling bounds with an extended state space.

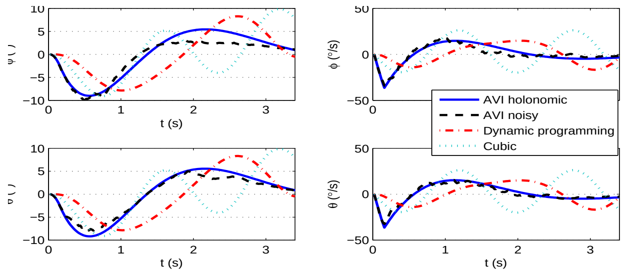
Table 1 presents the averaged results with their standard deviations. We measure the end state and the time when the agent reaches the goal, the percentage of trajectories that reach the goal state within 15 seconds, and the maximum swing experienced among all 100 trials. With the exception of the noisy holonomic simulator at the starting position (-20,-20,15), all experiments complete the trajectory within 4 cm of the goal, and with a swing of less than $0.6°$, as Proposition 3.1 predicts. The trajectories using the noisy simulator from a distance of 32 meters (-20,-20,15) don't reach within 5 cm because 11% of the trajectories exceed the 15-second time limit before the agent reaches its destination. However, we still see that the swing is reduced and minimal at the destination approach, even in that case. The results show that trajectories generated with stochastic simulator on average take 5% longer to reach the goal state, and the standard deviations associated with the results is larger. This is expected, given the random nature of the noise. However, all of the stochastic trajectories approach the goal with about the same accuracy as the deterministic trajectories. This finding matches our prediction from Section 3.2.

The maximum angle of the load during its entire trajectory for all 100 trials depends on the inverse distance from the initial state to the goal state. For short trajectories within the sampling box, the swing always remains within $4°$, while for very long trajectories it could go up to $46°$. As seen in Figure 3, the peak angle is reached at the beginning of the trajectory during the initial acceleration, and as the trajectory proceeds, the swing reduces. This makes sense, given that the agent is minimizing the combination of the swing and distance. When very far away from the goal, the agent will move quickly toward the goal state and produce increased swing. Once the agent is closer to the goal state, the swing component becomes dominant in the value function, and the swing reduces.

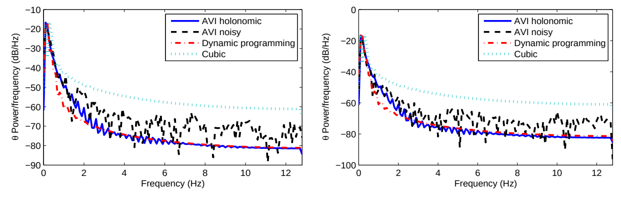
Figure 4 shows the comparison of the trajectories with the same starting position (-2, -2, 1) and the same $\mathbf{\Theta}$ parameter, generated using the models above (AVI trajectories) compared to cubic and DP trajectories. First, we see that the AVI trajectories share a similar velocity profile (Figure 4a) with two velocity peaks, both occurring in the first half of the flight. Velocities in DP and cubic trajectories have a single maximum in the second half of the trajectory. The resulting swing predictions (Figure 4b) shows that in the last 0.3 seconds of the trajectory, the cubic trajectory exhibits a swing of $10°$, while the DP trajectory ends with a swing of less than $5°$. The AVI generated trajectories produce load displacement within $2°$ in the same time period. To assess energy of the load’s motion and compare different trajectories in that way, Figure 4 shows one-sided power spectral density of the load displacement angles. We see that the energy per frequency of the cubic trajectory is above the other three trajectories. Inspecting the average energy of AVI holonomic ($E([\phi\ \theta]) = [0.0074\ 0.0073]$) , noisy AVI ($E([\phi\ \theta]) = [0.0050\ 0.0050]$), and DP trajectories ($E([\phi\ \theta]) = [0.0081\ 0.0081)$ load position signals, we find that AVI holonomic trajectory requires the least amount of energy over the entire trajectory.



(a) Quadrotor position and velocity



(b) Load position and velocity



(c) Power spectral density

Figure 4: AVI trajectory comparison to DP and cubic trajectories in simulation. Trajectories of the (a) vehicle, (b) its load, and (c) load displacement’s power spectral density where the training was performed in 3D configuration, and the trajectories were generated using generic and noisy holonomic simulators compared to the cubic and dynamic programming trajectories of the same duration.

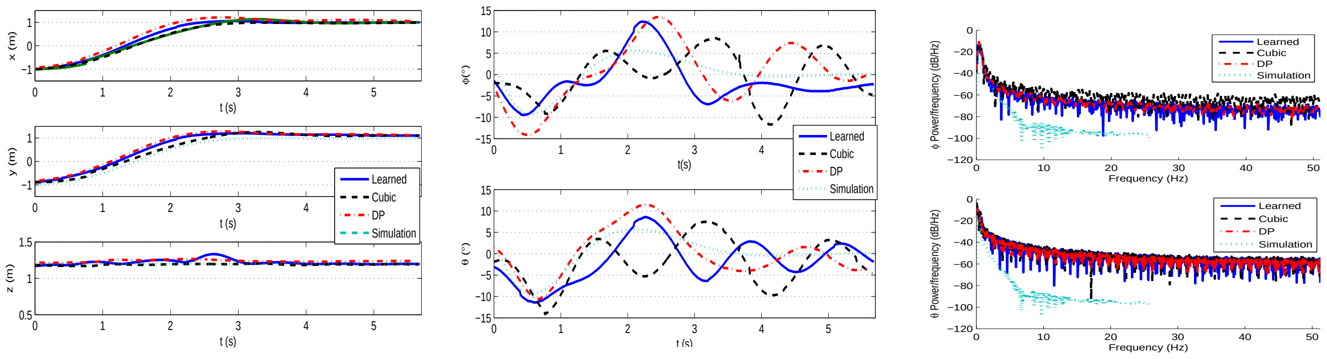
4.2.2 Experimental results

We first trained an agent in 2D configuration. The 2D configuration uses a finer discretization of the action space, although only in the x and y directions. There are 121 actions in each direction, totaling to 121^2 actions in the discretized space. This configuration uses a fixed sampling methodology. [5] show that the approximation error stabilizes to a roughly constant level after the parameters stabilize. Once the agent was trained, we generated trajectories for two experiments: constant altitude flight and flight with changing altitude. To generate trajectories, we used a fine-grain, discretized 3D action space $A = (-3 : 0.05 : 3)^3$. Trajectories were generated at 50Hz using the generic holonomic aerial vehicle carrying a suspended load simulator, the same simulator that was used in the learning phase. These trajectories were sent to the quadrotor with a suspended load weighing 45 grams on a 62 cm-long suspension cable.

In the constant altitude flight, the quadrotor flew from (-1,-1,1) to (1,1,1). Figure 5 compares the vehicle and load trajectories for a learned trajectory as flown and in simulation, with cubic and DP trajectories

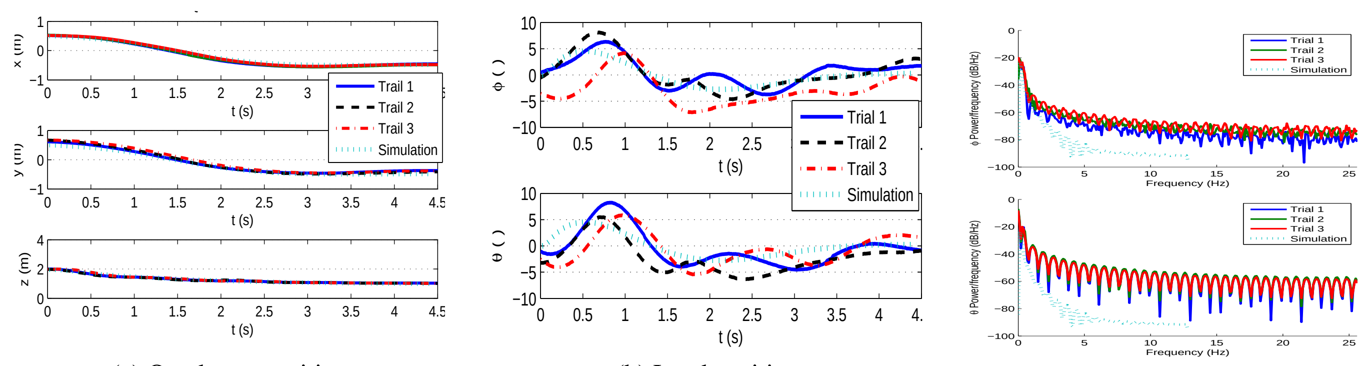
Table 1: Summary of AVI trajectory results for different starting position averaged over 100 trials: percent completed trajectories within 15 seconds, time to reach the goal, final distance to goal, final swing, and maximum swing.

State		Goal reached	t (s)		$\parallel \boldsymbol{p} \parallel (m)$		$\parallel \boldsymbol{\eta} \parallel (^\circ)$		max $\parallel \boldsymbol{\eta} \parallel (^\circ)$	
Location	Simulator	(%)	μ	σ	μ	σ	μ	σ	μ	σ
(-2,-2,1)	Deterministic	100	6.13	0.82	0.03	0.01	0.54	0.28	12.19	1.16
	Stochastic	100	6.39	0.98	0.04	0.01	0.55	0.30	12.66	1.89
(-20,-20,15)	Deterministic	99	10.94	1.15	0.04	0.01	0.49	0.33	46.28	3.90
	Stochastic	89	12.04	1.91	0.08	0.22	0.47	0.45	44.39	7.22
((4,5),(4,5),(4,5))	Deterministic	100	7.89	0.87	0.04	0.01	0.36	0.31	26.51	2.84
	Stochastic	100	7.96	1.11	0.04	0.01	0.44	0.29	27.70	3.94
((-1,1),(-1,1),(-1,1))	Deterministic	100	4.55	0.89	0.04	0.01	0.33	0.30	3.36	1.39
	Stochastic	100	4.55	1.03	0.04	0.01	0.38	0.29	3.46	1.52



(a) Quadrotor position (b) Load position (c) Power spectral density

Figure 5: AVI experimental trajectories. Quadrotor (a), load (b) trajectories, and power spectral density (c) as flown, created through learning compared to cubic, dynamic programming, and simulated trajectories.



(a) Quadrotor position (b) Load position (c) Power spectral density

Figure 6: AVI experimental results of altitude changing flight. Quadrotor (a), load (b) trajectories, and power spectral density (c) as flown and in simulation, over three trials in the altitude changing test trained in planar action space.

of the same length and duration. The vehicle trajectories in Figure 5a suggest a difference in the velocity profile, with the learned trajectory producing a slightly steeper acceleration between 1 and 2.5 seconds. The learned trajectory also contains a 10 cm vertical move up toward the end of the flight. To compare the flown trajectory with the simulated trajectory, we look at the load trajectories in Figure 5b. We notice the reduced swing, especially in the second half of the load's ϕ coordinate. The trajectory in simulation never exceeds $10°$, and the actual flown trajectory reaches its maximum at $12°$. Both learned load trajectories follow the same profile with three distinct peaks around 0.5 seconds, 2.2 seconds, and 3.1 seconds into the flight, followed by rapid swing control and reduction to under $5°$. The actual flown trajectory naturally contains more oscillations that the simulator didn't model. Despite that, the limits, boundaries, and profiles of the load trajectories are close between the simulation and flown trajectories. This verifies the validity of the simulation results: the load trajectory predictions in the simulator are reasonably accurate. Comparing the flown learned trajectory with a cubic trajectory, we see a different swing profile. The cubic load trajectory has higher oscillation, four peaks within 3.5 seconds of flight, compared to three peaks for the learned trajectory. The maximum peak of the cubic trajectory is 14° at the beginning of the flight. The most notable difference happens after the destination is reached during the hover (after 3.5 seconds in Figure 5b. In this part of the trajectory, the cubic trajectory shows a load swing of $5 - 12°$, while the learned trajectory controls the swing to under $4°$. Figure 5b shows that the load of the trajectory learned with reinforcement learning stays within the load trajectory generated using dynamic programming at all times: during the flight (the first 3.4 seconds) and the residual oscillation after the flight. Power spectral density in Figure 5a shows that, during experiments, learned trajectory contains less energy per frequency than cubic and DP trajectories.

In the second set of experiments, the same agent was used to generate changing altitude trajectories that demonstrate ability to expand action space between learning and planning. Note that the trajectories generated for this experiment used value approximator parameters learned on a 2D action space, in the xy-plane, and produced a viable trajectory that changes altitude because the trajectory generation phase used 3D action space. This property was predicted by Proposition 3.1 since the extended 3D action space allows transitions to the higher value states. The experiment was performed three times and the resulting quadrotor and load trajectories are depicted in Figure 6. The trajectories are consistent between three trials and follow closely simulated trajectory (Figure 6a). The load displacement (Figure 6b) remains under $10°$ and exhibits minimal residual oscillations. Figure 6c shows that the energy profile between the trials remains consistent.

4.3 Swing-free path tracking

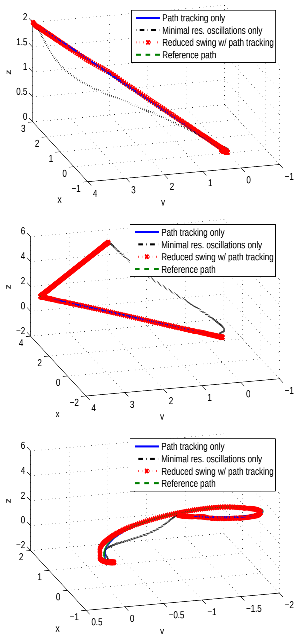
Path tracking with reduced load displacement evaluation compares the load displacement and tracking errors of the Algorithm 1 with two other methods: a minimal residual oscillations method described in Section 3.2, and path tracking with no load displacement reduction. The path tracking-only method chooses actions that transition the system as close as possible to the tracking trajectory in the direction of the goal state with no consideration to the load swing. We use three reference trajectories: a straight line, a two-line segment, and a helix. To generate the trajectory, we use action space discretized in 0.1 equidistant steps, and the same value function parametrization, $\mathbf{\Theta}$, used in the evaluations in Section 4.2 and learned in Section 3.1.

Table 2: Summary of path tracking results for different trajectory geometries: reference path, m, proximity factor (δ), trajectory duration, maximum swing, and maximum deviation from the reference path (error). Best results for reference path are highlighted.

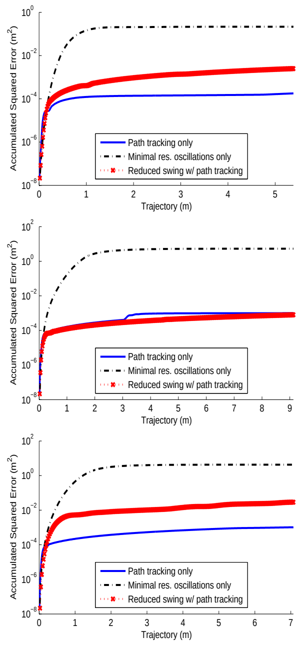
Ref. path	m	δ (m)	t (s)	$\|\|\eta\|\|$ $(^\circ)$	Error (m)
Line	**100**	**0.01**	**11.02**	**21.94**	**0.02**
	500	0.01	11.02	20.30	0.03
	100	0.05	6.64	23.21	0.08
	500	0.05	7.06	23.22	0.11
Tracking only	100	0.01	11.02	73.54	0.01
Minimal residual oscillations	-	-	6.74	18.20	0.49
Multi-segment line	**100**	**0.01**	**7.52**	**20.17**	**0.04**
	500	0.01	7.52	23.11	0.50
	100	0.05	7.52	28.24	0.10
	500	0.05	7.52	26.70	0.88
Tracking only	100	0.01	11.02	44.76	0.04
Minimal residual oscillations	-	-	6.74	16.15	2.19
Helix	100	0.01	5.72	29.86	0.39
	500	0.01	5.72	26.11	0.44
	100	0.05	5.76	32.13	0.17
	500	**0.05**	**5.72**	**22.20**	**0.11**
Tracking only	100	0.01	11.02	46.01	0.02
Minimal residual oscillations	-	-	7.68	4.30	1.80

Table 2 examines the role of the proximity parameter δ, and candidate actions set size parameter m. Recall that the Algorithm 1 used δ as a distance from the reference trajectory where all actions that transition the system within δ distance will be considered for load swing reduction. If there were no such actions, then the algorithm selects m actions that transition the system the closest to the reference trajectory, regardless of the actual physical distance. We look at two proximity factors and two action set size parameters. In all cases, the proposed method, swing-free path tracking, exhibits smaller path tracking error than minimal residual oscillations method, and smaller load displacement results than tracking-only method. Also for each reference path, there is a set of parameters that provides good balance between tracking error and load displacement reduction.

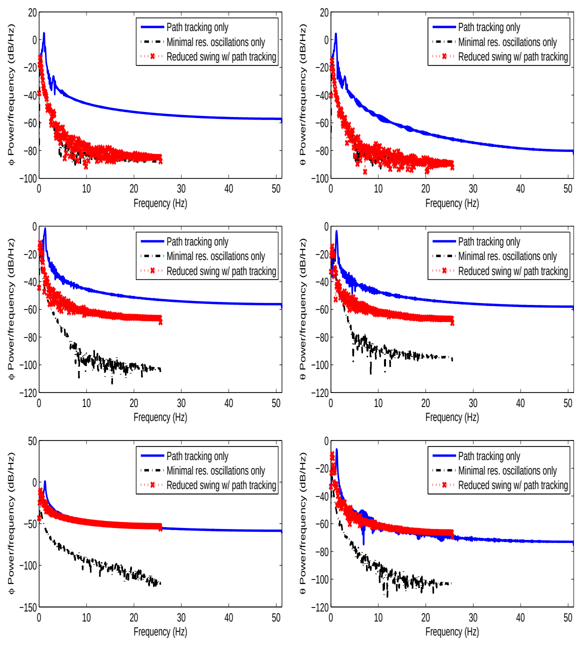
Figure 7 presents results of the tracking and load displacement errors for the same three reference paths: line, multi-segment, and helix. We compare them to tracking-only and minimal residual oscillations AVI algorithms. Figure 7a displays the trajectories. We see that in all three cases, the minimal residual oscillations algorithm took a significantly different path from the other two. Its goal was to minimize the distance to the goal state as soon as possible while reducing the swing, and it does not follow the reference



(a) Trajectory



(b) Line tracking error



(c) Power spectral density

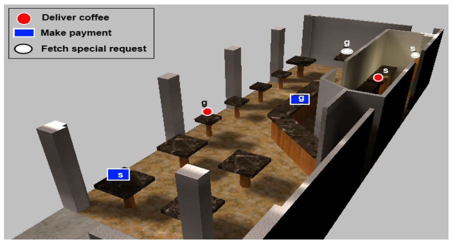
Figure 7: Swing-free path tracking for line, multi-segment, and helix reference trajectories, compared to path tracking only, and to load displacement control only (a). Tracking error (b) is in logarithmic scale, and power spectral density (c).

path. Figure 7b quantifies the accumulated tracking error. Path tracking error for Algorithm 1 remains close to the tracking-only trajectory. For all three trajectories, the accumulated tracking error is one to two orders of magnitude smaller than for the minimal residual oscillations method. Examining the load displacement characteristics though power spectral analysis of the vector $\boldsymbol{\eta} = [\phi\ \theta]^T$ time series, we notice that frequency profile of the trajectory generated with Algorithm 1 resembles closely that of the minimal residual oscillations trajectory (Figure 7c). In contrast, power spectral density of tracking only trajectories contain high frequencies absent in the trajectories created with the other two methods. Thus, the proposed method for swing-free path tracking, with its tracking error characteristics similar to the tracking-only method, and its load displacement characteristics similar to the minimal residual oscillations method, offers a solid compromise between the two extremes.

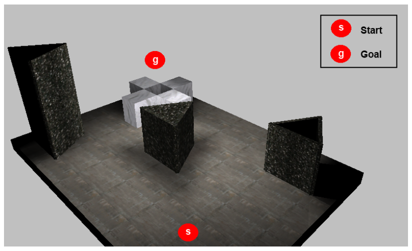
4.4 Automated aerial cargo delivery

In this section we evaluate the methods for an aerial cargo delivery task developed in 3.5. Our goal is to verify that the method finds collision-free paths and creates trajectories that closely follow the paths while not exceeding the given maximal load displacement. The method's performance in simulation is discussed in Section 4.4.1, and its experimental evaluation, in Section 4.4.2.

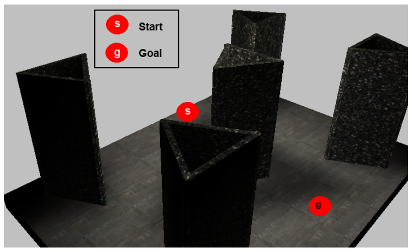
The simulations and experiments were performed using the same setup as for minimal residual oscillations tasks in obstacle-free environments, described in Section 4.2. In our PRM set up, we use uniform random sampling. Each node in the roadmap is connected to its 10 nearest neighbors using Euclidean distance. A straight line planner with a resolution of 5 cm creates the edges in the graph. The path planning was done using the Parasol Motion Planning Library from Texas A&M University [22].



(a) Cafe



(b) Testbed Environment 1



(c) Testbed Environment 2

Figure 8: Benchmark environments studied in simulation (a) and experimentally (b-c)

4.4.1 Trajectory planning in Cafe

To explore conceptual applications of the quadrotors to domestic and assistive robotics and to test the method in a more challenging environment, we choose a virtual coffee shop setting for our simulation testing. In this setting, the cargo delivery tasks (Problem 2.2) include: delivering coffee, delivering checks to the counter, and fetching items from high shelves (see Figure 8a). The UAV needs to pass a doorway, change altitude, and navigate between the tables, shelves, and counters. In all these tasks, both speed and load displacement are important factors. We want the service to be in a timely manner, but it is important that the swing of the coffee cup, which represents the load, is minimized so that the drink is not spilled. The Cafe is 30 meters long, 10 meters wide, and 5 meters tall.

We generate the paths, and create trajectories for three different maximal load displacements (1°, 10°, and 25°). Since the path planning is the most complex for the largest bounding volume, and the same path can be reused for smaller maximal load displacements, we create the path once using the bounding volume with a 25° cone half aperture, and create trajectories along the path requiring the same or lower maximal load displacement. The proximity factor is $\delta = 5\ cm$ and the candidate action sets size is $m = 500$. We evaluate the number of added waypoints to meet the load displacement requirement, the trajectory duration, the maximum swing, and the maximum path-tracking error.

Table 3 summarizes the results of the trajectory characteristics for the three tasks in the coffee shop for different maximal load displacement. It demonstrates that we can control load and follow the path to an arbitrary small load displacement. The maximum swing along a trajectory always stays under the required load displacement bound. The tracking error stays within 10 cm, regardless of the path and decreases as the maximal load displacement bound decreases. Therefore, by designing the bounding volume with 10 cm of clearance, the quadrotor-load system will not collide with the environment. The delivery time and number of added waypoints increase with the decrease of the required load displacement bound, as expected. Although, it is common sense that slower trajectories produce less swing, the agent automatically chooses waypoints so not to needlessly slow down the trajectories.

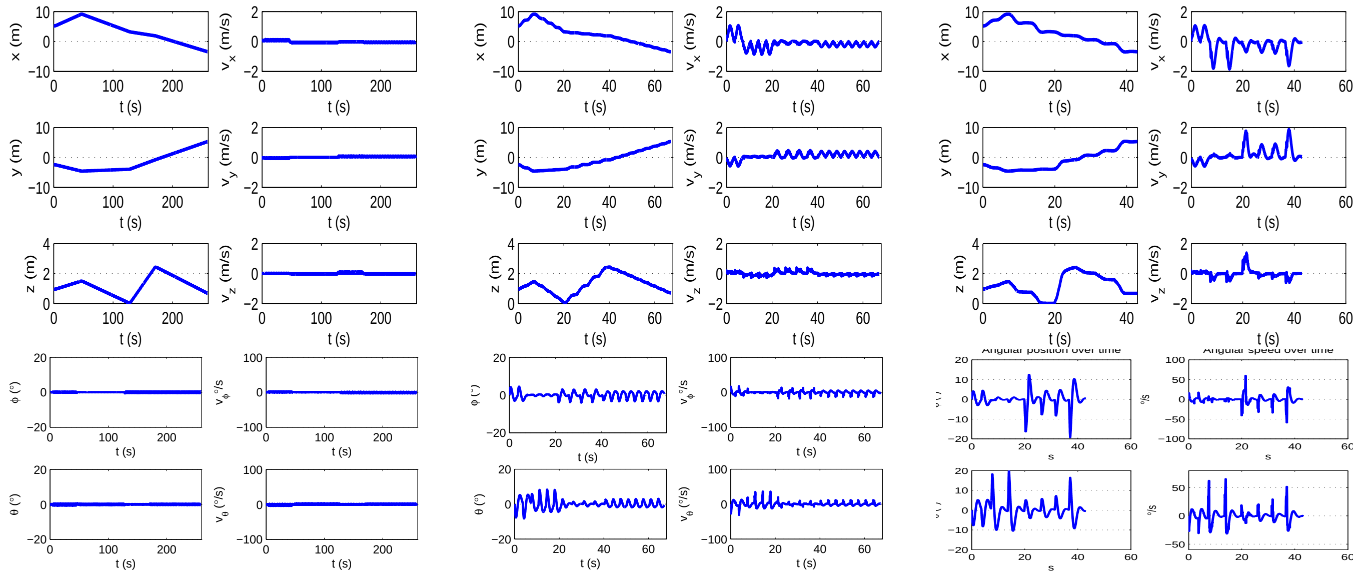
Figure 9 depicts the trajectory of the quadrotor and the load during the coffee delivery task for required maximal load displacements of 1°, 10°, and 25°. The trajectories are smooth, and the load's residual oscillations are minimal in all of them. The trajectory and path overlay is presented in Figure 10. The number of inserted waypoints increases for the smaller angles. The tracking error over the trajectory length is displayed in Figure 10d. The accumulated squared error profiles differ based on the load displacement angle bound, with the 1° trajectory having accumulated error significantly smaller than other two trajectories.

4.4.2 Experimental evaluation

The goals of the experiments are to show the discrepancy between the simulation results and the observed experimental trajectories, and to demonstrate the safety and feasibility of the method by using a quadrotor to deliver a cup of water. To check the discrepancy between the simulation predictions of the maximum

Table 3: Summary of path and trajectory results for different tasks in the Cafe setting: task name and maximum allowed load displacement ($\|\eta\|$), obstacle-free path length, and number of waypoints, trajectory waypoints after bisection, trajectory durations (t), maximum swing ($\|\eta\|$), and maximum deviation from the path (error).

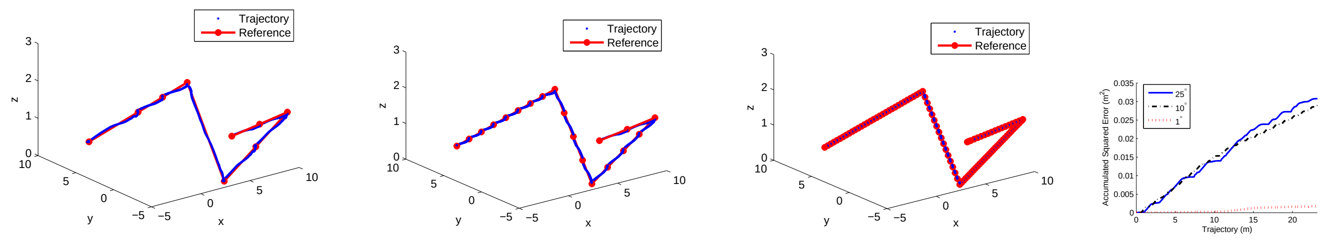
Task		Path		Trajectory			
Name	$\|\eta\|$ (°)	Length (m)	Pts.	Pts.	t (s)	$\|\eta\|$ (°)	Error (m)
Deliver coffee	25	23.20	3	7	43.02	24.60	0.08
	10	23.20	3	17	67.18	9.34	0.06
	1	23.20	3	97	258.62	0.86	0.03
Pay	25	15.21	1	3	25.98	18.23	0.06
	10	15.21	1	7	28.64	8.94	0.05
	1	15.21	1	63	172.28	0.92	0.01
Special request	25	32.57	1	7	52.54	24.11	0.07
	10	32.57	1	22	82.76	9.58	0.05
	1	32.57	1	128	349.86	0.92	0.01



(a) $\sigma = 1°$ (b) $\sigma = 10°$ (c) $\sigma = 25°$

Figure 9: Quadrotor and load trajectories in coffee delivery task in the Cafe for maximal allowed load displacement of 1°, 10°, and 25°. Zero altitude is the lowest point that a cargo equipped quadrotor can fly without the load touching the ground.

load displacement in simulation and experimentally, we run queries in two testbed environments to generate trajectories in simulation. The testbed environments are all 2.5 meters by 3 meters, with the ceiling height varying from 1.5 meters to 2.5 meters. The obstacles are uniform triangular prisms with 60 cm sides. Shorter obstacles are 60 cm tall, while the tall ones are 1.2 meters tall. They differ in number of obstacles, their sizes, and locations. The first testbed environment contains three obstacles positioned diagonally across the room and the same landing platform (see Figure 8b). Two obstacles are 0.6 meters tall, while the third one is 1.2 meters tall. The second testbed environment is filled with five 1.2 meters tall obstacles. They are in the corners and in the middle of a 2.5 by 2 meters rectangle centered in the room (Figure 8c). This is a challenging, cluttered space that allows us to experimentally test an urban environment setting.



(a) 25° (b) 10° (c) 1° (d) Tracking error

Figure 10: Path bisecting for Deliver coffee task based on the maximum allowed load displacement (a-c). Accumulated squared tracking error along the trajectory for different maximum allowed load displacements (d).

Table 4: Summary of path and simulated and experimental trajectory characteristics for different obstacle configurations:task name and maximum allowed load displacement (η); obstacle-free path length (l), and its number of waypoints (#); simulated trajectory: waypoints after bisection (#), planned trajectory durations (t), maximum swing (η), and maximum deviation from the path (error); experimental trajectory: maximum swing (η) and maximum deviation from the path (error). The experimental results are average over three trials.

Task		Path		Simulation				Experiment	
Test.	$\|\eta\|$ (°)	l (m)	#	#	t (s)	$\|\eta\|$ (°)	Error (m)	$\|\eta\|$ (°)	Error (m)
Env. 1	10	3.86	3	3	12.64	7.25	0.05	11.32	0.16
	5	3.86	3	4	15.68	3.63	0.03	7.99	0.11
	1	3.86	3	16	44.98	0.98	0.04	5.05	0.07
Env. 2	10	3.23	2	2	10.18	5.51	0.05	9.94	0.16
	5	3.23	2	4	15.80	2.66	0.05	6.72	0.11
	1	3.23	2	16	42.24	0.69	0.02	4.98	0.07

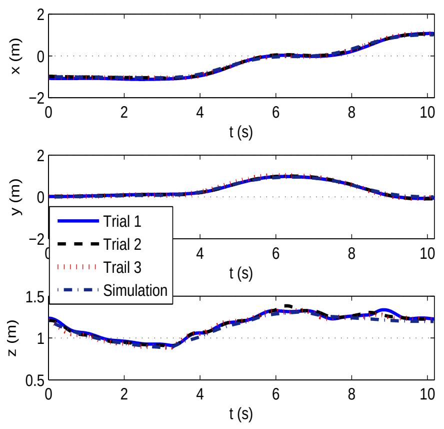
Table 4 summarizes the difference in observed versus predicted maximum load displacement for these tasks over three trials. The observed maximum displacement is between $4°$ an $5°$ higher experimentally than in simulation. This is expected and due to unmodeled system dynamics, noise, wind influence and other factors and it matches load displacement observed during hover.

Figure 11 shows three trials of the experimentally flown trajectory in Environment 2 (Figure 8c), with the predicted simulation. The vehicle trajectory matches very closely between trials and the simulation. The load’s trajectories show higher uncertainty over the position of the load at any given time. However, the load displacement is bounded and stays within $10°$.

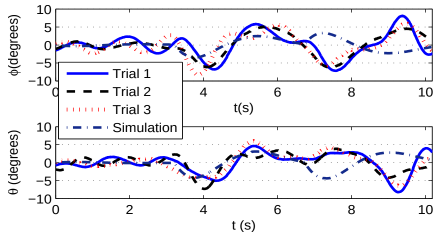
The demonstration of the practical feasibility and safety of this method was demonstrated by using the system to deliver a cup of water to a static human subject. In this demonstration, the quadrotor’s load is a 250 ml paper cup filled with 100 ml of water. In Environment 1 (see Figure 8b), a quadrotor needs to fly diagonally through the room, avoiding a set of three obstacles. The path and trajectory used for this demonstration are the same referenced in Table 4. A human subject was seated at the table. As the quadrotor competed the flight, it set the cup of water in front of the human, who detached it from the quadrotor. A video of the human-quadrotor interaction can be found in the enclosed video submission.

5 Conclusion

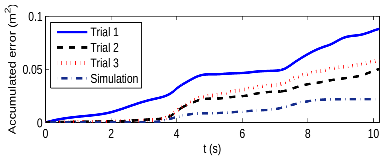
In this work we proposed an autonomous aerial cargo delivery agent that works in environments with static obstacles to plan and create trajectories with bounded load displacements. At the heart of the method is the reinforcement learning policy for minimal residual oscillations trajectories. This article showcases how a



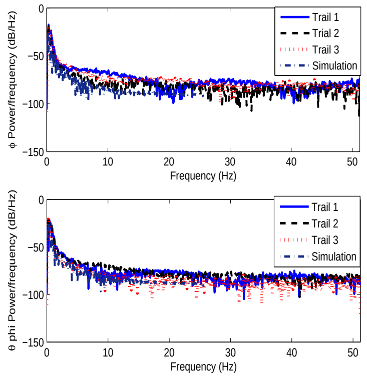
(a) Quadrotor position



(b) Load position



(d) Accumulated tracking error



(c) Power spectral density

Figure 11: Experimental quadrotor (a) and load (b) trajectories, power spectral density (c), and accumulated path tracking error (d) in the second testbed configuration.

reinforcement learning policy learned through a single training can be adapted to perform different tasks and impose nonholonomic constraints on the system. We modified the policy by restricting the domain's action space to track a reference path with a bounded load displacement. Finally, we integrated the swing-free path tracking and sampling-based path planning to solve the aerial cargo delivery task. We evaluated each of the modules separately, and showed that learning converges to a single policy, and that performs minimal residual oscillation delivery task, that the policy is viable with expending state and action spaces. The simulations quality is assessed through the comparison with experimental load displacement on a physical robot. Then we evaluated the swing-free path tracking on three reference paths for varying values of the tacking parameters. The results demonstrated that the proposed method attains both good tracking and good load displacement characteristics. Lastly, results of the integration with sampling-based motion planning shows that the method creates collision-free trajectories with bound load displacement for arbitrarily small load displacement bounds. We experimentally demonstrated the feasibility and safety of the method by having a quadrotor deliver a cup of water to a human subject.

This article lays a foundation for aerial cargo delivery in environments with static obstacles. In further work, trajectory smoothing can be applied to the trajectories generated with this to speed them up. Moving obstacles can be handled through an online trajectory tracking that adapts to dynamical changes in environment.

6 Acknowledgments

The authors would like to thank Torin Adamson for creating virtual models of the Cafe, the testbed and the UAV carrying the suspended load.

References

[1] AscTec. Ascending Technologies GmbH, 2013. http://www.asctec.de/.

[2] R. E. Bellman. *Dynamic Programming*. Dover Publications, Incorporated, 1957.

[3] M. Bernard and K. Kondak. Generic slung load transportation system using small size helicopters. In *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*, pages 3258 –3264, may 2009.

[4] D. Ernst, M. Glavic, P. Geurts, and L. Wehenkel. Approximate value iteration in the reinforcement learning context. application to electrical power system control. *International Journal of Emerging Electric Power Systems*, 3(1):1066.1–1066.37, 2005. Download from: http://www.bepress.com/ijeeps/vol3/iss1/art1066.

[5] A. M. Farahmand, R. Munos, and C. Szepesvári. Error propagation for approximate policy and value iteration. In *Advances in Neural Information Processing Systems*, 2010.

[6] A. Faust, I. Palunko, P. Cruz, R. Feirro, and L. Tapia. Learning swing-free trajectories for uavs with a suspended load. In *IEEE International Conference on Robotics and Automation (ICRA), Karlsruhe, Germany*, pages 4887–4894, May 2013.

[7] D. Hsu, J.-C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. In *ICRA*, pages 2719–2726, 1997.

[8] L. E. Kavraki, P. Švestka, J. C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robot. Automat.*, 12(4):566–580, August 1996.

[9] J. Kuffner, J.J. and S. LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, volume 2, pages 995 –1001 vol.2, 2000.

[10] V. Kumar and N. Michael. Opportunities and challenges with autonomous micro aerial vehicles. *The International Journal of Robotics Research*, 31(11):1279–1291, Sept. 2012.

[11] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006.

[12] S. M. Lavalle, J. J. Kuffner, and Jr. Rapidly-exploring random trees: Progress and prospects. In *Algorithmic and Computational Robotics: New Directions*, pages 293–308, 2000.

[13] S. Lupashin and R. DAndrea. Adaptive open-loop aerobatic maneuvers for quadrocopters. In *World Congress*, volume 18, pages 2600–2606, 2011.

[14] N. Malone, K. Manavi, J. Wood, and L. Tapia. Construction and use of roadmaps that incoroporate worksapce modeling errors. In *To appear IEEE International Conference on Intellignet Robots and Systems (IROS)*, Nov 2013.

[15] K. Manavi, B. S. Wilson, and L. Tapia. Simulation and analysis of antibody aggregation on cell surfaces using motion planning and graph analysis. In *ACM Conference on Bioinformatics, Computational Biology and Biomedicine*, October 2012.

[16] MARHES. Multi-agent, robotics, hybrid, and embedded systems laboratory, department of copmuter and electrical engineering, university of new mexico. http://marhes.ece.unm.edu, February 2013.

[17] H. B. McMahan, M. Likhachev, and G. J. Gordon. Bounded real-time dynamic programming: Rtdp with monotone upper bounds and performance guarantees. In *In ICML05*, pages 569–576, 2005.

[18] D. Mellinger, N. Michael, and V. Kumar. Trajectory generation and control for precise aggressive maneuvers with quadrotors. *The International Journal of Robotics Research*, 31(5):664–674, 2012.

[19] N. Michael, D. Mellinger, Q. Lindsey, and V. Kumar. The grasp multiple micro-uav testbed. *Robotics Automation Magazine, IEEE*, 17(3):56 –65, sept. 2010.

[20] I. Palunko, P. Cruz, and R. Fierro. Agile load transportation : Safe and efficient load manipulation with aerial robots. *Robotics Automation Magazine, IEEE*, 19(3):69 –79, sept. 2012.

[21] I. Palunko, A. Faust, P. Cruz, L. Tapia, and R. Feirro. A reinforcement learning approach towards autonomous suspended load manipulation using aerial robots. In *IEEE International Conference on Robotics and Automation (ICRA), Karlsruhe, Germany*, pages 4881–4886, May 2013.

[22] Parasol. Parasol lab, department of computer science and engineering, texas a&m university. https://parasol.tamu.edu/, December 2012.

[23] A. P. Schoellig, F. L. Mueller, and R. DAndrea. Optimization-based iterative learning for precise quadrocopter trajectory tracking. *Autonomous Robots*, 33:103–127, 2012.

[24] S. Shen, N. Michael, and V. Kumar. 3D estimation and control for autonomous flight with constrained computation. In *In Proc. IEEE Int. Conf. Robot. Autom. (ICRA),*, Shanghai, China, may 2011. Submitted.

[25] A. A. Sherstov and P. Stone. Improving action selection in MDP's via knowledge transfer. In *Proceedings of the Twentieth National Conference on Artificial Intelligence*, July 2005.

[26] K. Sreenath, N. Michael, and V. Kumar. Trajectory generation and control of a quadrotor with a cable-suspended load – a differentially-flat hybrid system. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4873–4880, 2013.

[27] M. E. Taylor and P. Stone. Behavior transfer for value-function-based reinforcement learning. In F. Dignum, V. Dignum, S. Koenig, S. Kraus, M. P. Singh, and M. Wooldridge, editors, *The Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 53–59, New York, NY, July 2005. ACM Press.

[28] Vicon. Vicon Motion Systems Limited, 2012. http://www.vicon.com/.